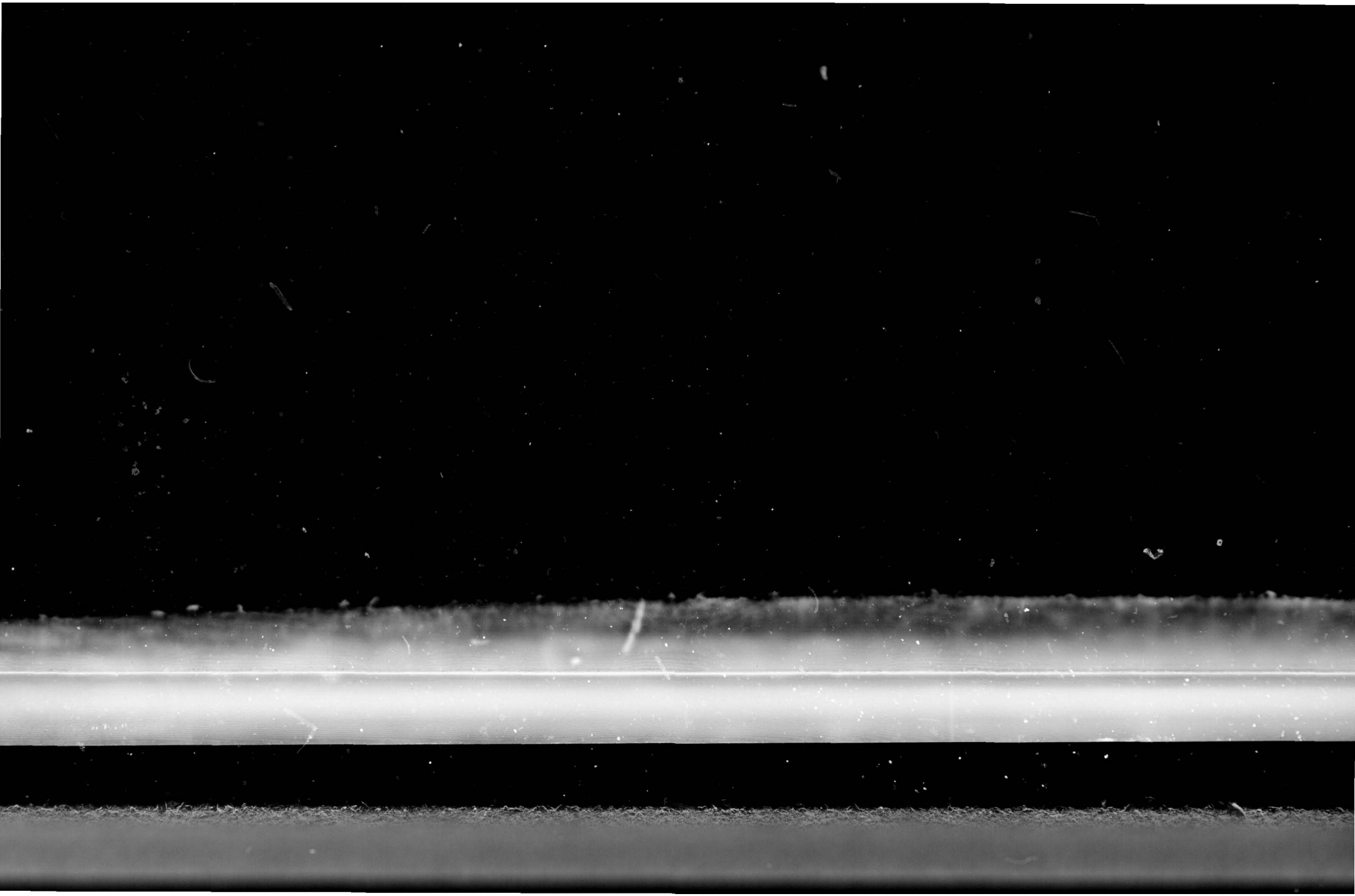
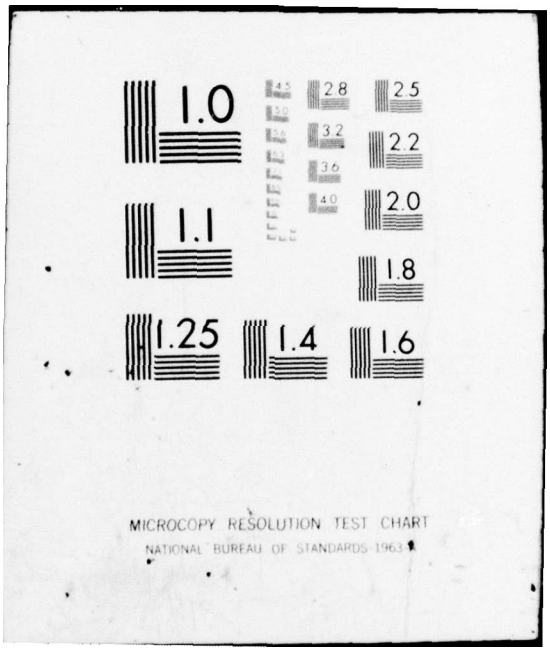
AD-A039 738 FEDERAL COBOL COMPILER TESTING SERVICE WASHINGTON D C F/G 9/2
SYSTEM FOR EFFICIENT PROGRAM PORTABILITY,(U)
MAY 77 G N BAIRD, L A JOHNSON

UNCLASSIFIED FCCTS/TR-77/08 NL

1 OF 1
AD
A039738

END
DATE
FILMED
6-77




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

System for efficient program portability

by GEORGE N. BAIRD and L. ARNOLD JOHNSON

Department of the Navy
Washington, D.C.

BACKGROUND

The acquisition of major Automatic Data Processing Equipment (ADPE) systems in the Department of the Navy is accomplished by either single source acquisition or competitive selection.[1] In the case of the single source acquisition, only one vendor is considered as having the system capable of satisfying the needs of the procuring organization. The criteria which must be satisfied for a single source acquisition include the need for unique hardware, excessive conversion/reprogramming cost and/or time, etc. The competitive selection, on the other hand, is open to any vendor who feels he can provide a system meeting the requirements specified in the solicitation document associated with that procurement.

The major steps in the competitive process are, briefly, as follows:

(1) The requestor prepares a solicitation document which explains the requirements of the system being procured.
(2) Benchmark programs are prepared. These will be used to ascertain that vendors participating in the competition can meet minimal performance standards. These programs may be written in various higher level languages. We will restrict our discussions to COBOL programs.
(3) The vendor examines the solicitation document to determine whether he has a system capable of meeting the requirements.
(4) The vendor obtains the benchmark, converts it to the system being considered, and determines if the hardware/software will be price-competitive with the systems most likely to be offered by his competitors.
(5) The vendor must then demonstrate that the execution of the benchmark can be accomplished within the time specified in the solicitation document.
(6) The award is generally made to the vendor who qualifies with respect to timed benchmarks and offers to provide the system which meets the needs of the user at the lowest overall cost to the Government.

The usual amount of time involved in the competitive selection process, as described above, ranges from nine to 23 months. A significant portion of this time is involved in working with the benchmark. The system we are describing here is designed to reduce this time.

USE OF BENCHMARKS IN THE SELECTION PROCESS

The benchmark is a vital part of the competitive selection process. It becomes the tool for minimum measurement to be used against all systems being considered. Therefore it is important that the benchmark be constructed in such a way as to accurately reflect the system requirements being specified. The system requirements are defined in terms of the current workload and the projected future workload. Properly prepared benchmarks will demonstrate that the system being offered contains adequate memory and peripherals, and that the throughput speeds are sufficient to process the projected future workload. Additionally, this exercise demonstrates that the operating system and supporting software are operative. From a functional standpoint, the "ideal benchmark" situation could be described as follows:

(1) The programs would be coded using the language elements defined in the American National Standard COBOL, so that source code conversion is minimal.
(2) The implementation of the benchmark programs by the various vendors could be monitored as in a controlled environment. This would provide useful information as to the impact of converting other programs after the new system is delivered.
(3) The programs have been debugged to the extent that they will give predictable results on both the native and the target machines.
(4) The data files would be in a form readily acceptable to all systems, but would at the same time be consistent with any given system's architecture, so that there is no loss of efficiency, or validity of results.
(5) The checking of benchmark processing results would be as automated as possible.

Unfortunately, this ideal situation seldom exists. This

See 1473

results in excessive time and cost expended on the part of the vendor in processing the benchmark. It is not unusual for a vendor to spend six to nine calendar months just preparing the benchmark programs for processing, or for the cost of processing them to represent 10 percent or more of the eventual bid price. The cost and time related to processing a benchmark causes vendors to be more selective in responding to requests for proposals (RFPs). This could result in the best system not being offered if the vendor feels he may not have a good chance of winning.

PREVIOUS BENCHMARK EFFORTS

Little has been done in the area of making the benchmark, and, as a result, the competitive process, more palatable. In the past, various methods have been used in presenting the benchmark to the vendor. These range from the attitude of "here is the benchmark, do with it what you must in order to make it run on your system, and in the meantime don't bother me" to a recent instance where the benchmark and its related data were provided in what could be described as "machine interchangeable" form (i.e., all data was in DISPLAY form, several "dangerous" language features were not used, etc.). This approach is consistent with recommendations made by Meltzer and Ickes.[2]

The effect of the "don't care" philosophy is that the vendor is permitted to make any changes he desires in implementing the benchmark. This may destroy the representativeness of the benchmark. At the same time, through the use of his most talented programmers and/or analysts, a vendor could optimize the programs for faster execution. It is often the case that the talent the vendor is able to turn loose on the benchmark is superior to that of the average user organization. Therefore, the credibility of the benchmark is somewhat lessened, and the timing results now represent more what the vendor's programming staff can do than what was originally intended.

The philosophy behind the machine interchangeable COBOL calls for no modification being permitted to the source programs except in the Environment Division. Basically, the idea of machine interchangeable COBOL is to eliminate any form of data representation except for standard data format (e.g., character representation only—no binary, packed decimal, floating point, etc.).

The machine interchangeable approach satisfies the majority of the criteria described in presenting the ideal benchmark; however there are two comments worth making:

(1) The decision to stay with the standard data format may force several vendors not to participate; primarily those with systems not capable of character addressing and/or decimal arithmetic.
(2) The resources used in manually converting the benchmark package to machine interchangeable format will include a substantial amount of manpower and computer time.

THE SANITATION EFFORT BY THE U.S. NAVY

The Software Development Division of the Department of the Navy Automatic Data Processing Equipment Selection Office (ADPESO) is looking into methods and techniques for decreasing the amount of time required for competitive selection, and lowering the overall cost of the procurement. Armed with the knowledge of problems associated with benchmark techniques, and in particular the problems associated with natural benchmarks, an effort was established to mechanize the preparation of natural benchmarks.

The vehicle we chose (for ease of implementation, with the necessary documented controls) was the VP-Routine developed by the Navy for automatically resolving implementation names within COBOL programs, and generating the necessary operating system control cards to compile and execute the programs.[3]

Each source program must be purged of nonstandard language elements. This is accomplished by automatic conversion (where possible through simple syntax replacement) and hand tailoring when additional logic may be required to handle semantic differences between two statements. Also, all implementor names must be resolved, or converted to an intermediate form. This results in COBOL source programs that are VP-Routine sensitive in that they can be tailored to a given systems requirements by a single pass of the VP-Routine.

All data files necessary for input to benchmark programs, or output files provided for verification purposes, must be carefully extracted from the native system and transformed into a form readily acceptable to other systems. This must be done with no loss of data integrity.

After the source programs and data have been converted, it is important to insure that the execution of the converted benchmark programs give the same results as the original programs, and that the data has not been adversely affected. This is accomplished by executing the sanitized benchmark using the sanitized data on both the current computer system and on other target systems. During the execution of the benchmark it is useful to be able to determine the degree to which the data is exercising the various procedures of each program. Through the help of a software monitor, information is provided which will help to further determine the adequacy of the benchmark.

The result of the benchmark preparation process would be a viable product that is:

(1) Well documented.
(2) Checked out on more than one system.
(3) Easily implementable through the use of the VP-Routine.

A discussion of the overall system follows in five major sections:

Source Program Preparation
Data Portability
Program Monitor
Packaging and Distribution
Limitations

SOURCE PROGRAM PREPARATION

The source programs that are to make up the benchmark are processed by a COBOL to COBOL translator that performs three major functions.

(1) The Environment Division is rendered VP-Routine sensitive by the replacement of all implementor names with an encoded mnemonic that has meaning to the VP-Routine. These mnemonics are later used in preparing the programs for a given system.
(2) The source code is examined for nonstandard coding, and translated when appropriate. Where translation cannot be accomplished, the translator flags the offending code.
(3) Based on parameter cards, the file descriptions of the files to be converted from the native system are used to create an intermediate work file containing pseudo file/record/field descriptions. This file is later used to create Data Translation programs. This is fully discussed in the section entitled Data Portability.

The translation of one COBOL dialect to another is conceptually simple. The more serious problem involves the moving of data from system to system. A detailed discussion of this problem and a suggested solution follows.

DATA PORTABILITY

Problem

The differences in the internal representation of data among computer systems represent the major limitation of software portability. These differences can be broadly segmented into two categories: Differences in the character code used (i.e., EBCDIC, BCD, FIELDATA, etc.), and differences in the representation of numeric data. Data translators for character code conversion are widely available, or can be easily created. Transferability of the second type requires more than simple code conversions, and, therefore, presents the greater problem. Furthermore, the specific representation used within this general category will, for a given computer, seriously impact the effectiveness with which that computer is used. Thus, the system described here concerns itself only with the conversion of noncharacter data from any "native" computer to any "target" computer, in such a way that the target computer architecture is properly utilized.

Generally, COBOL data portability is impacted by variations in numeric data representation, alignment of data within a defined data unit (e.g., a word), and the position and representation of arithmetic signs. There are several forms of numeric data representation, of which binary and packed decimal are the most common. The packed decimal format is not universal and may therefore have to be converted to a completely different type of data structure. Data in binary representation *are* universal, and although sign conventions and word size do vary, conversion between binary representations is relatively simple. Alignment variations affect the positioning of data within storage units, particularly in word-oriented computers. For proper transformation of data from external storage to internal storage, the COBOL program definition of this data must be consistent with the expected data position and alignment on the external files.

Solution

Generality and *impartiality* are principal design goals of our effort. Our system must perform data translation from *any* given system to any other system. Furthermore, we must not penalize the architecture of the target system. Generality implies that the translation programs must be automatically generated, as opposed to hand-coded. Impartiality means we must go from a machine dependent form to another machine dependent form. Good sense suggests we do this through an intermediate machine independent code.

We use available software as much as possible. This is done by using the code conversion subroutines already present in a system's compiler, together with the data descriptions in the COBOL programs being converted. The data translators which constitute the heart of the system are automatically generated COBOL program segments. These data translators are used to convert native machine dependent data (MDD) to standard data format (SDF), and the latter to target machine dependent data, which we refer to as machine ANSI data (MAD). If character code translation is required, it can be performed on the SDF, since this data is simply a string of characters.

Program creation

Data translation/verification programs (henceforth referred to simply as *data translators*) are created from the file/record descriptions of the COBOL programs being converted. The data translators will contain the following COBOL file descriptions (FD's):

(1) Machine dependent data (MDD) file descriptions, which are those used to process the file on the native machine.
(2) Standard data format (SDF) file descriptions, in which all data items are in DISPLAY mode, unsigned and unsynchronized.
(3) Machine ANSI data (MAD) file descriptions, in which all data items are described in Standard COBOL formats.[4]
(4) Machine ANSI data for the target machine (MADT) file descriptions. This file description is identical to (3) above, and is used for file comparison purposes. This comparison process is more fully described below.

Source code MDD item descriptions which are not Standard COBOL will be defined by the MAD file description in a form which is as close to the native file description as possible (e.g., COMPUTATIONAL-3 will become COMPU-

```
        01 MDD-RECORD.
           02 ALPHANUMERIC-D    PICTURE X(20)    JUSTIFIED RIGHT.
MDD        02 UNSIGNED-D       PICTURE 9(6)     COMPUTATIONAL-3
           02 SIGNED-D         PICTURE S9(6).

        01 SDF-RECORD.
           02 ALPHANUMERIC-X    PICTURE X(20).
SDF        02 UNSIGNED-X       PICTURE 9(6).
           02 SIGNED-S          PICTURE X.
           02 SIGNED-X          PICTURE 9(6).

        01 MAD-RECORD.
           02 ALPHANUMERIC-A    PICTURE X(20)    JUSTIFIED RIGHT.
MAD        02 UNSIGNED-A       PICTURE 9(6)     COMPUTATIONAL
           02 SIGNED-A          PICTURE S9(6).
```

Figure 1—Example of record description used in a data translator program

TATIONAL). Figure 1 illustrates a DATA DIVISION from a data translation program. Procedures for translating from one data form to another and for file comparisons (for data verification purposes) are generated for each elementary field of the record. There are three data translation procedure types, corresponding to alphanumeric data, signed numeric data, and unsigned numeric data. The type of procedure generated is based on the elementary COBOL item description.

Data translation procedures for alphanumeric and unsigned numeric data require no more than a COBOL MOVE statement. Any changes in the data code, data alignment, or storage allocation required in converting from one form to another are performed automatically by the code generated (for the MOVE statement) by the compiler being used.

To take into account the various sign conventions, COBOL procedures are added to check the characteristics of signed numeric data. If translation is from a machine dependent form to a SDF form, the appropriate sign is stored as a separate character in the SDF data description. If we are performing the reverse process, we first generate the positive value of the machine dependent data item (through a MOVE statement), then check the separate sign character in the SDF description, and if minus multiply the machine dependent data item by minus one to give it the correct sign.

Data validation and verification procedures are also generated. Figures 2 and 3 give examples of the various COBOL procedures used for data translation, validation, and verification (the latter two functions are described below).

Once all the procedures and file descriptions have been generated, they are merged with appropriate housekeeping COBOL statements, resulting in data translators which are complete COBOL programs in VP-Routine sensitive format.

Program operation

Since the native source code file descriptions in the data translators may not be acceptable on the target compiler, the VP-Routine is used to provide the capability of selecting the appropriate source coding for use on the desired system (target or native). This is accomplished by parameter cards to the VP-Routine. If any minor updating to the source programs is required, this capability is also available through the VP-Routine.

Parameters are used as input to the data translators in order to direct the flow of execution. The three categories of functions which may be performed are data translation, data verification, and data validation.

Four modes of translation are available. The mode required

```
                    MOVE ALPHNUMERIC-D TO ALPHNUMERIC-X.
                    MOVE UNSIGNED-D TO UNSIGNED-X.
                    IF  SIGNED-D NEGATIVE
(MDD to SDF             MOVE "-" TO SIGNED-S
procedure)              ELSE
                        MOVE "+" TO SIGNED-S.
                    MOVE SIGNED-D TO SIGNED-X.

                    MOVE ALPHNUMERIC-X TO ALPHNUMERIC-A
                    MOVE UNSIGNED-X TO UNSIGNED-A.
(SDF to MAD         MOVE SIGNED-X to SIGNED-A.
procedure)          IF  SIGNED-S EQUAL TO "-"
                        MULTIPLY -1 BY SIGNED-A.
```

Figure 2—Example of data translation procedures used in a Data Translator Program

```
                    IF  UNSIGNED-X NUMERIC NEXT SENTENCE ELSE
                        MOVE  UNSIGNED-X TO PRT-FIELD-DATA
(Data Validation        MOVE  "UNSIGNED-X" TO PRT-FIELD-NAME
procedure)              PERFORM PRINT-VALIDATION-ERROR.

                    IF  UNSIGNED-A NOT EQUAL TO UNSIGNED-D
                        MOVE  UNSIGNED-A TO FLDA-NUMERIC-18V
                        MOVE  UNSIGNED-A TO FLDA-NUMERIC-V18
                        MOVE  "UNSIGNED-A" TO FIELD-NAME
(Data Verification      MOVE  UNSIGNED-D TO FLDB-NUMERIC-18V
procedure)              MOVE  UNSIGNED-D TO FLDB-NUMERIC-V18
                        PERFORM  PRINT-VERIFICATION-ERROR.
```

Figure 3—Example of data validation and data verification procedures used in a data translator program

is indicated by the following parameter cards:

```
CONVERT MDD-SDF
CONVERT SDF-MAD
CONVERT MDD-MAD
CONVERT MAD-SDF
```

The first mode is applicable to the native compiler, and translates Machine Dependent Data to Standard Data Format. The second mode is applicable to the target compiler, and translates Standard Data Format to Machine ANSI Data. The last two data translation modes are used to create files for data verification.

There are three modes of data verification. The specific one required is indicated by one of the following parameter cards:

```
COMPARE MDD-MAD
COMPARE MAD-MADT
COMPARE SDF-MAD
```

The first mode of data verification is used to compare Machine ANSI Data files to Machine Dependent Data files. The second mode is used to compare two Machine ANSI Data files. The last mode is mainly for flexibility, and performs a SDF to MAD translation before a MAD to MADT comparison is made.

The third function performed by the conversion system is data validation. This consists of verifying that the data content is consistent with its class characteristics (i.e., numerically described fields should contain only numeric data). This function is performed automatically in combination with the translation function, or during a separate pass, using a VALIDATE SDF-DATA parameter.

Data validation/verification

In order to perform a comparative evaluation of the performance of computer systems through the use of natural benchmarks, we must ensure that the same amount of processing was completed by all competing systems, and that the accuracy of computations is within allowable bounds. Data comparison and validation procedures are included in our system for this purpose. Additionally, these procedures provide the benchmark recipient with a tool to check the various stages of a multi-step processing application for any processing inconsistencies. Finally, the procedures are used to confirm that data integrity is not lost in either the program conversion or data conversion process.

Processing integrity is verified in two ways through the authentication of data files and comparisons of files after program execution. The authentication of data files consists of validating the data item content for conformance to their described data class (i.e., numeric fields contain the data values 0 through 9 and, possibly, a sign). This specific feature was incorporated in the system because it has been found, for example, that some compiler implementations permit spaces as data in numeric fields, or maintain signed data in fields described as unsigned, and provide the appropriate translation before processing; other implementations do not. Validation would point out these potential problem areas.

The comparison of files after program execution provides a means of determining not only that all the processing was done, but also that the numerical results of this processing are within the accuracy limits allowed. When any data discrepancy is found by the data translators, a report of the discrepancy is produced. A report is made for each field in error, and includes the name of the field as defined in the program, its data content (in the case of a comparison, the field being compared to and the comparing field are both displayed), the position of the field in the record, and relative record position in the file. Record and error counts are also provided in the report. Figure 4 gives an example of the report generated.

PROGRAM MONITOR

One of the principal concerns in using benchmarks as a means of evaluating computer performance is whether they provide an adequate representation of the user's workload, and whether they properly reflect his future processing needs. This problem is not completely resolvable, but an indication

DATA VALIDATION/VERIFICATION REPORT

LOGICAL RECORD	FIELD NAME	STARTING POSITION	FIELD SIZE	DATA CONTENTS
000006	UNSIGNED-A	0021	0006	+000000000000000443.00000000000000000000 (INCORRECT)
				•
				+000000000000000444.00000000000000000000 (CORRECT)
000020	ALPHNUMERIC-A	0001	0020	aaaaaaaacdeaaagjaaaa
				••• ••
				aaaaaaaaaaaaaaaaaaaa

MACHINE ANSI RECORDS = 004562
MACHINE DEPENDENT RECORDS = 004562
ERROR COUNT = 0002

Figure 4—Sample validation/verification report from a data translator program

of the processing characteristics of the programs provided for the benchmark can be of value in the evaluation process. This data is obtained through a program execution monitor.

Following program and data conversion, and before distribution of the benchmark to the vendors, this monitor is applied to the benchmark programs. The monitor, written in COBOL, inserts control statements into the benchmark programs. Execution of the benchmark programs with a given set of data provides a histogram of procedure activity in the programs. This, in turn, can be used to determine the suitability of the benchmarks in representing the user workload.

PACKAGING AND DISTRIBUTION

Once the benchmark has been sanitized and run on the native system to be assured that processing integrity has been maintained, the benchmark package is prepared for distribution. The package includes a source program library, benchmark data, and documentation.

The source library (population file) will contain the benchmark programs, data translator programs, the VP-Routine, and the operating system control language for the major computer systems. The VP-Routine selects the programs from the population file, transforms VP-Routine sensitive programs to machine dependent programs by satisfying all implementor defined names in the source program, and prepares the job control stream for submission to the operating system.

Data files for the benchmarks are distributed to the vendor on magnetic tape, in SDF format.

Documentation pertaining to the programs, data, instructions for implementation on a users system, and all information necessary to run the benchmark for a live test demonstration is included in the package. This includes the following information:

(1) Cross reference to data files by reel number.
(2) Cross reference to data files by program.
(3) Cross reference to data files for file name.
(4) Detailed instructions for implementation of the Data Translator/Verification Programs.
(5) Instructions for use of the VP-Routine.[3]
(6) A workload processing statement, which is a table providing a summary of all the pertinent information for implementation of the benchmark.
(7) Instructions and sample program for the extraction of the VP-Routine from the population file.
(8) Benchmark instructions.
(9) Individual program documentation, including any known areas which may cause implementation problems.
(10) For variable length records, or multiply defined records, a complete COBOL record description is given, or a record layout is provided together with its record type characteristics.
(11) A system flowchart of the benchmark.
(12) Listings of each program.

LIMITATIONS

Even though the Benchmark Preparation System resolves many of the difficulties involved in program and data portability, there are areas in which reprogramming will be required for complete conversion. The amount of reprogramming depends on the degree to which machine dependency has been imposed onto the program. Data that is not explicitly defined, or features for which ANSI Standard COBOL does not have a direct functional replacement cannot be detected by the sanitation process. The following are a few of the known programming, COBOL characteristics, or COBOL compiler implementation practices which have an impact on automation of the conversion process.

(1) *Functions in the native COBOL source program which cannot be directly replaced by features or elements of the ANSI language specification.* Such an example would be the READY TRACE statement or the TRANSFORM verb in IBM System/360 COBOL.[5] The ANSI language specification does not have an element or feature which directly performs these functions. To simulate this function requires manual conversion.

(2) *Incomplete or inadequate record descriptions.* This is due to describing fields or groups of fields as alphanumeric when their true descriptions could include other forms of data representation. An example of this technique on the IBM 360/370 would be a data field described as PICTURE X(4), when the data actually present should be defined as PIC S9(9) COMPUTATIONAL (binary), or a PIC X(2) definition of a data field which is in fact PIC S9(3) COMPUTATIONAL-3 (packed decimal). The above examples would not only cause the target compiler to incorrectly allocate storage but also would not provide the appropriate conversion processing, since the data is described as alphanumeric.

(3) *Multiply defined records which have different data structures within each record, and do not have a means of distinguishing between records.* The data conversion process is capable of translating multiply defined records, but only if they can be identified.

(4) *Machine dependencies fixed into the COBOL program itself.* This would include such things as assuming the initial value of a data item, initializing numeric storage areas with alphanumeric literals representing a machine's internal sign, or using the character set to represent non-character machine data. An example would be:

```
WORKING-STORAGE SECTION.
77 SIGN-FIELD PIC S999.
77 X-FIELD REDEFINES SIGN-FIELD
   PIC XXX.
PROCEDURE DIVISION.
SECTION-NAME SECTION.
PARAGRAPH-L.
MOVE +123 to SIGN-FIELD
IF X-FIELD EQUAL TO '12C' GO TO—
```

Implementations which do not generate positive sign over-punches would require the procedure to be modified before the program would function correctly.

(5) *Collating sequence of fields containing alphanumeric data which are critical to program processing and which are not completely defined.* This problem is somewhat reduced, however, in that COBOL instructions which may be affected by collating sequence are flagged by the COBOL to COBOL translator.

CONCLUSIONS

The Benchmark Preparation System was developed to reduce the nonportability and expense of using natural benchmarks without losing the characteristics of the users workload in terms of processing efficiency and representation. The results we have obtained indicate that these objectives can be met.

Our current test bed is a Navy benchmark containing 38 COBOL programs consisting of some 60,000 lines of source code, and includes some 150 data files. The native system is an IBM 360/50 and the target machines are a UNIVAC 1108 and HIS 6050. These programs and data files have been successfully converted to both the UNIVAC 1108 and HIS 6050. Preparation of the benchmark programs, development of data translator/verification programs, and the packaging of these were done on a UNIVAC 1108. Approximately 96 percent of the changes made to the programs were handled by this system. The remaining changes (manual) were necessitated by extension features with no counterparts in the ANSI COBOL standard. Generation of the data translators and sanitation of the benchmark programs for packaging required approximately two computer runs and one man hour of effort per benchmark program. The effort required on each system to set up the VP-Routine, and cleanly compile the programs has been averaging one-tenth man hour per program. Character code translation posed no problem, as each system had job control card options for transliteration (i.e., EBCDIC to BCD on the IBM/360 and BCD to FIELDATA on the UNIVAC 1108 and IBMC code to HIS 6000 code).

Based on our efforts, we believe that portability can be achieved by an automated means without sacrificing the efficiency of a computer system.

REFERENCES

1. Department of the Navy, *Specification, Selection and Acquisition of Automatic Data Processing Equipment (ADPE)*, SECNAVINST 5236.1, December 17, 1971.
2. Ickes, Hubert F. and Herbert S. Meltzer, *Draft Tutorial on Interchangeable Data Files,"* ANSI Task Group X3.2F (1970).
3. Chief of Naval Operations, Information Systems Division (Op-91), *COBOL Compiler Validation System*, VP-Routine users guide, January 1973.
4. American National Standards Institute, Incorporated, *USA Standard COBOL*, X3.23-1968.
5. *IBM System/360 Operating System Full American National Standard COBOL*. GC 28-6396-2, IBM Corporation (1970).

BIBLIOGRAPHIC DATA SHEET	1. Report No. (14) FCCTS/TR-77/08	2.	3. Recipient's Accession No.
4. Title and Subtitle (6) System for Efficient Program Portability			5. Report Date
		(11)	6. 9 May 77 (12) 8p
7. Author(s) (10) George N. Baird L. Arnold Johnson			8. Performing Organization Rept No.
9. Performing Organization Name and Address Federal COBOL Compiler Testing Service Department of the Navy ADPE Selection Office Washington, D. C. 20376			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address ADPE Selection Office Department of the Navy Washington, D. C. 20376			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

This technical paper describes a software system (written in COBOL 68 - X3.23-1968) designed to ease the pains of converting COBOL programs and their related data to different systems or to different software environments within the same computer system. The approach taken is novel in that two processes take place. The first process is to convert the source programs to a system independent form of COBOL. As a result of examining the source program during the source program conversion phase, the information found in the File Section is used to produce data file translation - COBOL programs which can read the data file on the old system, produce a system independent data file, read the system independent data file on the new system and convert it to a file on the new system which will take advantage of the new architecture of the hardware/software. The source program conversion processor took this into account and the converted source program will be able to process the new file with little or no modification.

17. Key Words and Document Analysis. 17a. Descriptors

COBOL
Validation
Software
Audit Routines
Verifying
Compilers
Standards
Programming Languages

17b. Identifiers/Open-Ended Terms

CCVS

17c. COSATI Field/Group 09/02

408438

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited	UNCLASSIFIED	7
	20. Security Class (This Page) UNCLASSIFIED	22. Price